

3. The Pumping Lemma for Regular Languages

Contents

- Non-regular languages
- Pumping lemma for regular languages
- Applying the pumping lemma
- Decidability
- Chomsky's classification
- Closure properties

Non-Regular Languages

- Consider the language
 $L = \{\varepsilon, ab, aabb, aaabbb, \dots\}$
that is, $\mathbf{a^n b^n}$ for $n = 0, 1, 2, \dots$
- Every word in L consists of a certain number of 'a's followed by **the same number** of 'b's.
There is an infinite number of such words, one for each value of n .

Non-Regular Languages

- Now consider the language generated by the RE $\mathbf{a^*b^*}$.
- Every word in this language consists of some (possibly none) 'a's followed by some (possibly none) 'b's. It also has an infinite number of words, but it is clearly regular, since it is generated by a RE.
- Clearly $L \subset \mathbf{a^*b^*}$, but it is not exactly equivalent to it, for example aab is in $\mathbf{a^*b^*}$, but not L .
- Is L regular?

$a^n b^n$ is non-regular (informal)

- **Assume that L is a regular language.**
- Then there must exist some FSA that recognised it.
- By definition, this FSA would have to have a finite number of states.

$a^n b^n$ is non-regular (informal)

- **Assume that L is a regular language.**
- Then there must exist some FSA that recognised it.
- By definition, this FSA would have to have a **finite** number of states.
- Assume that the FSA that recognises L has, say, 95 states (or k).
- By definition, this FSA must accept all words in L and reject all words not in L .

$a^n b^n$ is non-regular (informal)

- Now, the string $a^{96}b^{96}$ is in L and the first 96 ($=k+1$) letters of this string are 'a'.
- Our FSA cannot visit a new state for each of the 'a' characters, since it has only 95 states (k states).

$a^n b^n$ is non-regular (informal)

- Now, the string $a^{96}b^{96}$ is in L and the first 96 ($=k+1$) letters of this string are 'a'.
- Our FSA cannot visit a new state for each of the 'a' characters, since it has only 95 states (k states).
- Therefore, at some point it must return to a state that it has visited previously, while still reading 'a's.
- If this happens, there is a **cycle** in the transition diagram and if there is such a cycle, then the FSA may loop round that cycle as many times as it likes, reading as many 'a's as it likes (consistently with the length of the cycle).

$a^n b^n$ is non-regular (informal)

- Say the cycle has 7 states (j states) in it, then the FSA would also accept words such as $a^{96+7}b^{96}$ ($a^{96+j}b^{96}$)
- But such a word is not in L because it has more 'a's than 'b's.

$a^n b^n$ is non-regular (informal)

- Say the cycle has 7 states (j states) in it, then the FSA would also accept words such as $a^{96+7}b^{96}$ ($a^{96+j}b^{96}$)
- But such a word is not in L because it has more 'a's than 'b's.
- So the FSA accepts some words not in L , that is, **it does not recognise L .**
- *Note that the choice of 95 for the number of states, and of 7 for the length of the cycle, have no effect on this argument, which may be constructed for arbitrary numbers.*

$a^n b^n$ is non-regular (informal)

- **This is a contradiction: any FSA that recognises L does not recognise L !**
- We must therefore **reject** our original assumption and conclude that
 - **L is not regular.**

$a^n b^n$ is non-regular (informal)

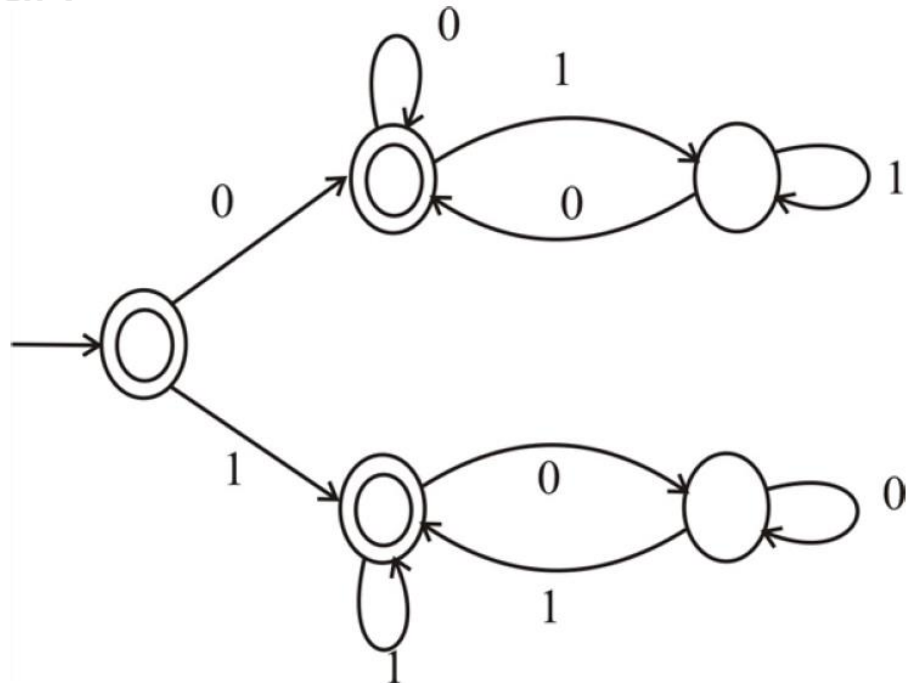
- Finite State Automata have a finite number of states.
- Since there are fewer states than strings, one state must be visited by two strings.
- After reading two different sequences a^i and a^j , the machine is in the same state q . After i a 's or j a 's the machine starts to read b 's, and it needs to remember whether before it read i or j a 's in order to accept if the number of b 's is the same. But the machine is in the same state q , so it cannot remember the number of a 's.

Regular vs non-regular

- Can we determine non-regular languages based on memory requirements?
- Consider the language $L = \{w \mid w \text{ contains an equal number of sequences } 01 \text{ and of } 10\}$, with $\Sigma = \{0,1\}$
- Is this language regular?

Regular vs non-regular

- Can we determine non-regular languages based on memory requirements?
- Consider the language $L = \{w \mid w \text{ contains an equal number of sequences } 01 \text{ and of } 10\}$, with $\Sigma = \{0,1\}$
- Is this language regular?



Pumping Lemma

- **Theorem (Pumping Lemma for Regular Languages):**
- If **A**: L is a regular language,
- Then **B**: there exists **a number k** (the pumping length) such that for **all words** w in L **whose length is at least k** , there exist strings x , y and z that divide w into a partition $w = xyz$ such that
 - 1: all strings of the form $xy^n z$ are in L , for $n \geq 0$ (including $n=0$)
 - 2: $|y| > 0$ (y has not zero length)
 - 3: $|x| + |y| \leq k$ (equivalently $|xy| \leq k$)
- (Note: $|x|$ denotes the length of the string x)

$A \Rightarrow B$, equivalently: $\text{not } B \Rightarrow \text{not } A$

Proof of the Pumping Lemma

Preliminary Construction:

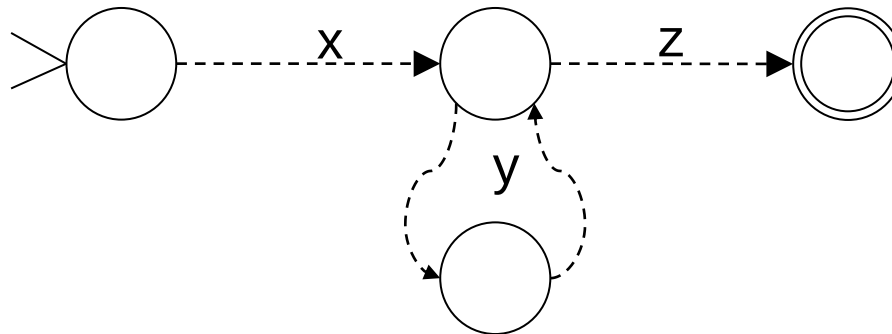
- If L is a regular language, then there is a (deterministic) FSA that accepts **exactly** (all and only) the words in L .
- This FSA has a finite number of states, but L has infinitely many words in it, so L must contain some words that are longer than the number of states of the FSA.

Proof of the Pumping Lemma

- Let the pumping length k be *the number of states* of the FSA. Let w be some word in L that is **at least as long as k** .
 - When this word generates a path through the machine, the path cannot visit a new state for each symbol because for this we would need at least $k+1$ states. Reminder: $k+1$ states are visited when reading k symbols (including both start and end state of the path)
 - Therefore, it must at some point revisit a state that it has been to before.

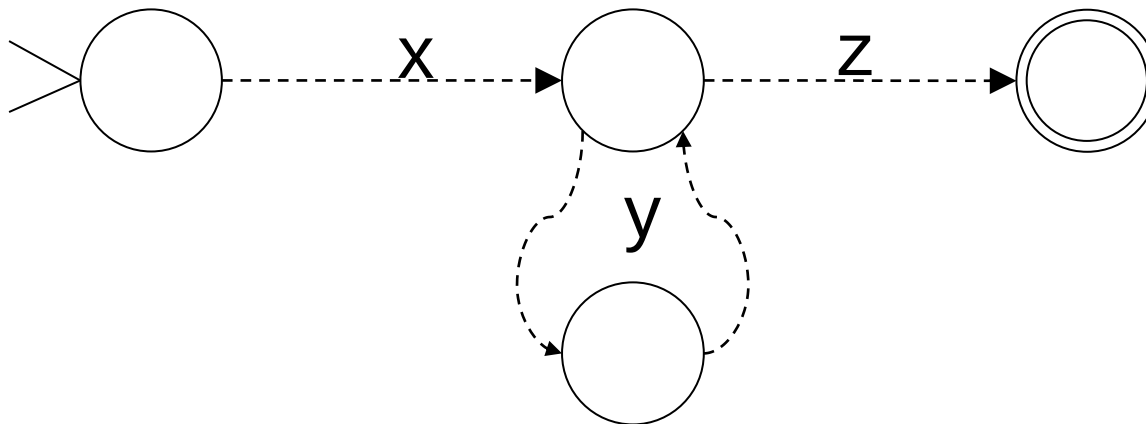
Proof of the Pumping Lemma

- Break the word **w** up into three parts.
 1. The prefix of **w** that leads to the first state that is revisited.
Call this part **x** (it could be null if the state revisited is the start state).
 2. Starting at the letter after **x**, let **y** denote the substring of **w** that travels around the circuit (cycle) exactly once, coming back to the state the circuit began with.
 - Since there must be a circuit, **y cannot be the null string** (cond. 1).
 - Also, note that $|x|+|y| \leq k$ (cond. 3) (as we have visited at most k states before the first repetition)



Proof of the Pumping Lemma

3. Starting at the letter after **y**, let **z** be the string that goes to the end of the string **w**.
- **z** could be null. The path for **z** could also loop around **y** (or any other circuit).



Proof of the Pumping Lemma

- Clearly, **every** word $w = xy^nz$ for $n = 0, 1, 2, 3, \dots$ is accepted by this FSA.
- The FSA recognises L , so every word accepted by it is in L .
- Therefore, all words of the form $w = xy^nz$ for $n = 0, 1, 2, 3, \dots$ are in L .
- This holds for every word w , with $|w| \geq k$ and for every infinite regular language.
- **End of Proof**

Pumping Lemma

- **Theorem (Pumping Lemma for Regular Languages):**
- If **A**: L is a regular language,
- Then **B**: there exists **a number k** (the pumping length) such that for **all words** w in L **whose length is at least k** , there exist strings x , y and z that divide w into a partition $w = xyz$ such that
 - 1: all strings of the form $xy^n z$ are in L , for $n \geq 0$ (including $n=0$)
 - 2: $|y| > 0$ (y has not zero length)
 - 3: $|x| + |y| \leq k$ (equivalently $|xy| \leq k$)
- (Note: $|x|$ denotes the length of the string x)

$A \Rightarrow B$, equivalently: $\text{not } B \Rightarrow \text{not } A$

Pumping Lemma

- **Theorem (Pumping Lemma for Regular Languages):**
- If **A**: L is a regular language,
- Then **B**: there exists a **number k** (the pumping length) such that for **all words** w in L **whose length is at least k** , there exist strings x , y and z that divide w into a partition $w = xyz$ such that
 - 1: all strings of the form $xy^n z$ are in L , for $n \geq 0$ (including $n=0$)
 - 2: $|y| > 0$ (y has not zero length)
 - 3: $|x| + |y| \leq k$ (equivalently $|xy| \leq k$)
- (Note: $|x|$ denotes the length of the string x)

$A \Rightarrow B$, equivalently: $\text{not } B \Rightarrow \text{not } A$

Pumping Lemma

- **Theorem (Pumping Lemma for Regular Languages):**
- If **A**: L is a regular language,
- Then **B**: there exists **a number k** (the pumping length) such that for **all words** w in L **whose length is at least k** , there exist strings x , y and z that divide w into a partition $w = xyz$ such that
 - 1: all strings of the form $xy^n z$ are in L , for $n \geq 0$ (**including $n=0$**)
 - 2: $|y| > 0$ (y has not zero length)
 - 3: $|x| + |y| \leq k$ (equivalently $|xy| \leq k$)
- (Note: $|x|$ denotes the length of the string x)

$A \Rightarrow B$, equivalently: $\text{not } B \Rightarrow \text{not } A$

Pumping Lemma

- **Theorem (Pumping Lemma for Regular Languages):**
- If **A**: L is a regular language,
- Then **B**: there exists a **number k** (the pumping length) such that for **all words** w in L **whose length is at least k** , there exist strings x , y and z that divide w into a partition $w = xyz$ such that
 - 1: all strings of the form $xy^n z$ are in L , for $n \geq 0$ (including $n=0$)
 - 2: $|y| > 0$ (y has not zero length)
 - 3: $|x| + |y| \leq k$ (equivalently $|xy| \leq k$)
- (Note: $|x|$ denotes the length of the string x)

Some non-regular languages fulfil the pumping lemma

$$A \Rightarrow B, \underline{\text{not}} A \Leftrightarrow B$$

Pumping Lemma

- **Theorem (Pumping Lemma for Regular Languages):**
- If **A**: L is a regular language,
- Then **B**: there exists **a number k** (the pumping length) such that for **all words** w in L **whose length is at least k** , there exist strings x , y and z that divide w into a partition $w = xyz$ such that
 - 1: all strings of the form $xy^n z$ are in L , for $n \geq 0$ (including $n=0$)
 - 2: $|y| > 0$ (y has not zero length)
 - 3: $|x| + |y| \leq k$ (equivalently $|xy| \leq k$)
- (Note: $|x|$ denotes the length of the string x)
- Any **finite language** fulfils the pumping lemma and even more, it is a regular language

Pumping Lemma

- **Theorem (Pumping Lemma for Regular Languages):**
- If **A**: L is a regular language,
- Then **B**: there exists **a number k** (the pumping length) such that for **all words** w in L **whose length is at least k** , there exist strings x , y and z that divide w into **a partition** $w = xyz$ such that
 - 1: all strings of the form $xy^n z$ are in L , for $n \geq 0$ (including $n=0$)
 - 2: $|y| > 0$ (y has not zero length)
 - 3: $|x| + |y| \leq k$ (equivalently $|xy| \leq k$)
- (Note: $|x|$ denotes the length of the string x)
- Any finite language fulfils the pumping lemma and even more, it is a regular language

Pumping Lemma

- Then **B**: there exists a number k (the pumping length) such that for all words w in L whose length is at least k , there exist strings x , y and z that divide w into a partition $w = xyz$ such that

.

Regular lang. $\Rightarrow$ Lemma fulfilled

Lemma Non-fulfilled $\Rightarrow$ Non-regular lang.

find one pumping length

discard all pumping lengths

lemma valid for all words $|w| \geq k$

for each k , find a word with $|w| \geq k$ for which it is not valid

find one partition for each word $|w| \geq k$

discard all partitions of the chosen words (one for each k)

Applying the Pumping Lemma

- In order to prove that a language L is **not regular**, first assume that it **is** (i.e. that some FSA can recognise it), then use the Pumping Lemma to reach a **contradiction**.
- This is known as **proof by contradiction**, (or **reductio ad absurdum**).
- The pumping lemma must be used in a proof that makes use of structural properties of the language being considered.
- Suitable words to be used as counterexamples should rely on the structural properties that reveal that pumping is not compatible with the language

Example

- Prove, using the Pumping Lemma, that the ‘addition’ language, $L = a^p b^q c^r$, where $p, q, r = 0, 1, 2, 3, \dots$ and $p + q = r$, is not regular.

Example

- Prove, using the Pumping Lemma, that the ‘addition’ language, $L = a^p b^q c^r$, where $p, q, r = 0, 1, 2, 3, \dots$ and $p + q = r$, is not regular.
- To apply the Pumping Lemma, **assume** that the language L is regular (so the lemma applies).
- Now demonstrate that this assumption leads to a contradiction.
- [Note: regularity implies the existence of an FSA with a finite number of states but which must recognise the infinite number of words in L , therefore it must have a cycle in it somewhere.]

Example: background thinking

- By the Pumping Lemma:
 - there exists a number k (the pumping length) such that:
 - For any word w with $|w| \geq k$ exists $w=xyz$, where
 - y is the string consumed by going round a cycle, with $|y| > 0$ and
 - x and z are any, possibly null, strings, with constraint $|xy| \leq k$
 - then L must also contain all strings of the form $xy^n z$, for $n = 0, 1, 2, 3, \dots$

Example: picking types of words

- The first thing that needs to be done is to choose a suitable type of words in the language. The pumping lemma applies to all words in the language equal and longer than the pumping length k , and **the key is to pick one type of words that in all possible cases of how y is defined will lead to a contradiction.**
- You can pick different words for different candidates of k , but for all k you must find one. This is why we consider how to choose **types** of words.

Structural Properties of 'Addition'

- Every non-null word of L is in the form
 $a...ab...bc...c$,
- that is:
 - A single (possibly null) string of 'a's followed by
 - A single (possibly null) string of 'b's, followed by
 - A single (possibly null) string of 'c's,
 - with no more than one point where an 'a' is immediately followed by a 'b'
 - and no more than one point where a 'b' is immediately followed by a 'c',
- where the number of 'a's plus the number of 'b's equals the number of 'c's.

Partitioning Analysis of 'Addition'

- There are exactly **6 different ways** (with respect to y) in which a word **$a...ab...bc...c$** (in L) may be partitioned into x , y and z :
 1. y is all 'a's
 2. y is some 'a's and some 'b's
 3. y is some 'b's
 4. y is some 'b's and some 'c's
 5. y is some 'c's
 6. y is some 'a's some 'b's and some 'c's
- The point being that you need to consider **all cases...**

Choosing types of words

- A good particular type of words will give you a simpler pumping argument in which you have to consider few cases (we just saw that for a generic word of the example language there are 6 cases to consider).
- For every k , the word chosen must have length at least k (in most cases will be much larger than k)

Choosing types of words

- A good particular type of words will give you a simpler pumping argument in which you have to consider few cases (we just saw that for a generic word of the example language there are 6 cases to consider).
- For every k , the word chosen must have length at least k (in most cases will be much larger than k)
- Recall that we must have $|x|+|y|\leq k$. This constraint helps to reduce the cases to consider:
- For each k , $a^k b^k c^{2k}$ is a word in the language.
- So, let us choose the word $a^k b^k c^{2k}$.

Choosing types of words and pumping

- The word is $w = xyz = a^k b^k c^{2k}$ and we have $|x| + |y| \leq k$.
- Therefore, y must be a non-null string of 'a's. (we have reduced the number of cases to one)
- You don't know how many, so describe it with an index: $y = a^j$, $0 < j \leq k$

Choosing types of words and pumping

- The word is $w = xyz = a^k b^k c^{2k}$ and we have $|x| + |y| \leq k$.
- Therefore, y must be a non-null string of 'a's. (we have reduced the number of cases to one)
- You don't know how many, so describe it with an index: $y = a^j$, $0 < j \leq k$ (because $|y| > 0$ and $|xy| \leq k$ are required)
 $x = a^i$, $y = a^j$, $z = \text{the rest}$, $0 \leq i < k$, $0 < j \leq k$, $i + j \leq k$

Choosing types of words and pumping

- The word is $w = xyz = a^k b^k c^{2k}$ and we have $|x| + |y| \leq k$.
- Therefore, y must be a non-null string of 'a's. (we have reduced the number of cases to one)
- You don't know how many, so describe it with an index: $y = a^j$, $0 < j \leq k$ (because $|y| > 0$ and $|xy| \leq k$ are required)
 $x = a^i$, $y = a^j$, $z = \text{the rest}$, $0 \leq i < k$, $0 < j \leq k$, $i + j \leq k$
- So, if word $w = xyz = a^k b^k c^{2k}$ is in language L
- then if language L is regular, by the pumping lemma $xyyz$ is also in L [this is the 'pumping' in the lemma].
- That is $a^{k+j} b^k c^{2k}$ is in language L .

Contradiction and conclusion

- However, observe that $a^{k+j}b^kc^{2k}$ is not in the language since $k + j + k \neq 2k$, as $j > 0$ (since $y = a^j$, and $|y| > 0$).
- Therefore, the use of the pumping lemma has led to a contradiction.
- The assumption that the language is regular must be incorrect, and it is concluded that the language is not regular.

End of proof.

Potential Pitfall

- Note that you must be careful that for your choice of words, **any possible choice of y** must lead to a contradiction.
- Sometimes this will mean have to consider several cases: one for each possible choice of y still consistent with $|x|+|y|\leq k$

Careful analysis

- The wrong choice of word can lead to difficulties in your argument.
- Consider, for example, the language **Palindrome** (e.g. mom, noon) over the alphabet $\{a,b\}$, which comprises every string that reads the same when written backwards.
- For example:
 - the null string is a word in **Palindrome**
 - as are 'a', 'b', 'aa', 'abba', 'aba', 'aabaa' etc.

Careful analysis

- Assume that Palindrome is regular, then the pumping lemma applies.
- Suppose k is the pumping length from the pumping lemma.
- Take the word ab^ka , which is in the language.
- In this set up, one possibility is that $y = b^j$ for some $j > 0$.
- However, in this case $xyyz = ab^{k+j}a$, which is in the language Palindrome.

Careful analysis

- This does not mean that Palindrome is regular, just that you've failed to show that it is not.
- We need a better choice of word (next slide).

Palindrome

- Suppose **Palindrome** is regular.
- The pumping lemma applies **for all words** longer than k .
- The word $w = a^{k+1}ba^{k+1}$ is a palindrome.
- Take $w = xyz = a^{k+1}ba^{k+1}$.
- Because $|x|+|y|$ must be less than or equal to k , both x and y must be strings of 'a's.

Palindrome

- So when the string $xxyz$ is formed, 'a's are added to the front of w but not to the back, because all the 'a's in the rear of w are in the z part, which stays fixed with $k+1$ 'a's.
- More precisely, if $y = a^j$, $0 < j \leq k$, then $xyyz = a^{k+1+j}ba^{k+1}$
- So the string $xxyz$ is not in **Palindrome**.
- But the Pumping Lemma requires **Palindrome** to include this string.
- Therefore, **Palindrome** is not a regular language.

Decidability and Chomsky's Classification

- Some questions that one can ask about FSAs and REs are:
- Equivalence:
 - **Do two given FSAs define the same language?**
 - **Do two given REs define the same language?**
- Finiteness:
 - **Is the language defined by a given FSA finite or infinite?**
- Membership:
 - **Is this word part of the language recognised by the FSA?**
- Emptiness:
 - **Is the language empty?** (equal to the empty set)
- These are yes/no questions.
- An effective solution to a yes/no question is called a **decision procedure**.

Decidability and Chomsky's Classification

- A problem that has a decision procedure is called **decidable**.
- It has been proven that all these questions are **decidable for the regular languages**, but this is not the case for all languages.
- Noam Chomsky introduced a **classification of languages** in which classes of language are distinguished according to the decidability of questions such as these, and according to closure properties.

Closure Properties

- Chomsky's classification also addresses questions of **closure**.
- That is, is the class of languages **closed under**
 - **union** (i.e. is the union of any two languages in the class also in that class)?
 - **intersection**?
 - **Product** (concatenation)?
 - The **product** of two languages, A and B, comprises the set of all strings that are formed by concatenating a word in A with a word in B.

Closure Properties

– **complement?**

- The **complement** of a language, L , whose alphabet is some set, S , of symbols, comprises the set of all strings that are in S^* but not in L . In other words, it is the set difference $S^* - L$.

– **Kleene Star?**

- The **Kleene star** of a language L is the set of all strings that can be written as the concatenation of zero or more strings from L .

The Chomsky Classification

Language Class	Type 3	Type 2	Type 1	Type 0
Known as	Regular	Context Free	Context Sensitive	Recursively Enumerable
Corresponding Classes of Acceptor Machine	Finite State Automata Transition Graph	Pushdown Automata	Linear Bounded Automata [TM with bounded tape]	Turing Machine
Deterministic = Non-Deterministic?	Yes: DFSA = NFSA	No. NPDA has more [DPDA = LR(k) grammars]	Yes	Yes, but...
Closed Under?	Union, Product, Intersection, Kleene Star, Complement	Union, Product, Kleene Star	Union, Product, Kleene Star	Union, Product, Intersection, Kleene Star
What's Decidable?	Equivalence, Emptiness, Finiteness, Membership	Emptiness, Finiteness, Membership	Membership	Not much!
Examples of Application	Text Editors, Sequential Circuit Synthesisers	Programming Language Statements, Compilers	Programming Languages, Compilers	Computers

Proof of language being non-regular by Using Closure

- Closure properties make some proofs of non-regularity much easier.
- Consider the language **Equal**, comprising of
 - all strings from the alphabet $\{a,b\}$ that contain equal numbers of 'a's and 'b's.
- Some of the words in **Equal** are:
 - the null string, 'ab', 'ba', 'aaabbbb', 'abab', and so on.
- It is difficult to see a suitable way of breaking a typical word of **Equal** into three parts in preparation for 'pumping'.

Proof of language being non-regular by Using Closure

- Instead, observe that

$$\{a^n b^n\} = \mathbf{Equal} \cap a^* b^*$$

- The language defined by the regular expression $a^* b^*$ is, of course, regular.
- If **Equal** were a regular language, then $\{a^n b^n\}$ would be the intersection of two regular languages and therefore, by the closure property of regular languages, it would have to be regular itself.
- But $\{a^n b^n\}$ is **not** regular, as shown earlier.
- So **Equal** cannot be regular.

End of Proof

Summary

- A language is either regular or not.
- One can try to show it is regular by finding a FSA that recognizes it.

Or,

- one can try to show it is not regular by assuming it is, applying the Pumping Lemma and arriving at a contradiction.

Reading

- From Sipser
- Section 1.4, pages 77-82.
- Closure (regular languages): pages 45-47, 59-62, 85
- Look at exercises 1.29-1.30.